DIJKSTRA ADVANCED PROGRAMMING GROUP PROJECT DOCUMENTATION

AUTHORS:
DIJKSTRA

CREATED ON: 23/03/2023

CONTENTS

1. PROBLEM DEFINITION
2. CLASSES AND FUNCTIONS
3. USER INTERFACE
4. OUTPUT
5. TESTING
6. REFERENCES

1. PROBLEM DEFINITION

Build a C++ program to apply a range of image filters and orthographic projections to an input 2D image or 3D data volume, using the programming techniques you have learned during the Advanced Programming course.
Your program should contain a main function which reads an image or data volume, asks the user which filter or projection to apply, calls separate functions for the appropriate filter and/or projections, and saves the new image with the filter/projection applied. Your program should work with any arbitrary image/data volume size.
You can use the existing libraries we have included in the src directory in your GitHub repository to read and write the image (stb_image.h and stb_image_write.h).

A minimal example showing how to include these header files in your code and how to read and write image files is included in the src/minimal.cpp file. You may also use C++ standard libraries and headers (e.g., string, iostream, cmath, vector). All other classes and functions in your code should be written by your group. No other external libraries may be used.

Code structure: Your code should include a main.cpp file containing the main method for your program. You should create classes called Image, Volume, Filter, Projection, and Slice. Each class should have a .cpp and .h file associated with it. You may create additional source code and header files for any other classes that you consider appropriate.

Your program must include at least six of the following ***2D filter types***, and at least two from each category. Each member of the group should write at least one image filter.
1) Color correction (simple per-pixel modifiers):
a. Grayscale (reduce image from 3 RGB channels to 1 grayscale channel)
b. Automatic color balance (adjust each RGB channel so the average intensity of each channel is equal)
c. Brightness (may take an optional brightness value in the range -255 – 255 to add to all pixels, or perform an automatic brightness filter by setting the average value of all pixels in all channels to 128)
d. Histogram equalization

2) Image blur using convolution filters (of different kernel sizes):
a. Median blur (replace each pixel value with the median of those around it), arbitrary kernel size (3x3, 5x5, 7x7, etc)
b. Box blur, arbitrary kernel size (3x3, 5x5, 7x7, etc)
c. Gaussian blur (5x5, 7x7 kernels)

3) Edge detection convolution filters. For these filters, you should first apply the grayscale filter from category 1 first, before applying the edge detection filter. You may find that edge detection filters work better on images which have been blurred with a box

or gaussian filter first; if so, document that in your report. You program could either output an image after each filter is applied or allow the user to specify multiple filters to use sequentially.
a. Sobel (3x3 operators)
b. Prewitt (3x3 operators)
c. Scharr (3x3 operators)
d. Roberts' Cross (2x2 operators)
Your program should also be able to read in a 3D data volume (e.g., a medical CT scan or an X-ray CT scan of porous media), optionally apply a 3D filter, and then perform an orthographic projection.

You should implement both of the following **3D filters**:
1) 3D Gaussian (to replace the center voxel with the weighted average of all the voxel intensities in the neighborhood), for an arbitrary filter size (e.g., 3x3x3, 5x5x5).

2) 3D Median (similar to the 2D median filter above, but for an arbitrary 3D kernel size, e.g., 3x3x3, 5x5x5).

The **orthographic projections** to implement are:
1) Maximum intensity projection (MIP) — find the maximum intensity of all pixels with a given (x, y) coordinate in all images in the z-direction, and output that maximum value on the final image.

2) Minimum intensity projection (MinIP) — like the MIP, but for the minimum intensity at each (x, y) coordinate.

3) Average intensity projection (AIP) — like the MIP, but for the average (mean or median) intensity of all pixels with a given (x, y) location.

2. FUNCTIONS AND CLASSES

The program consists of 6 classes. Their functions and properties are defined below.

2.1 *Image Class:*

Class "Image" represents an image in memory.The class has the following private Members.
width: an integer that represents the width of the image in pixels
height*:* an integer that represents the height of the image in pixels
channel: an integer that represents the number of color channels in the image
data: a pointer to an unsigned char array that holds the pixel data of the image

- **int Image::get_width()**
  Returns the width of the image as an integer

- **int Image::get_height()**
  Returns the height of the image as an integer

- **int Image::get_channel()**
  Returns the number of channels of the image as an integer

- **void Image::set_width(int w)**
  Sets the private member(width) of the class
  @params
  ---
  integer representing the width of the image
  @return

  None

- **void Image::set_height(int h)**
  Sets the private member(height) of the class
  @param
  ---
  integer representing the height of the image
  @return

  None

- **void Image::set_channel(int c)**
  Sets the private member(channel) of the class
  @params
  ---
  integer representing the number of channels
  @return

None

- **void Image::set_data(unsigned char *dat)**
  Sets the private member(data) of the class
  @param
  ---
  unsigned char * representing the image data

Example:

```
#include “Image.h”
int w, h,c;
unsigned char * data;
data = stbi_load(“path_to_image”, &w, &h, &c, 0);
Image img(w, h, c, data);
```

2.2 *Filter Class:*

Class "Filter" provides various image processing methods for color correction, image blur, and edge detection. The class has the following methods:

- **void Filter::grayscale(Image img**)
  Convert the given image to grayscale and save it to a new file.

  @param
  ---
  img: The input image object to be converted to grayscale.

  @return

  None.

  The function takes an Image object and calculates the grayscale value of each pixel using
  the formula gray = 0.299 * R + 0.587 * G + 0.114 * B. The resulting gray values are stored
  in a new unsigned char array. The number of channels in the input image is set to 1, and the grayscale image is saved to a new file named 'output_gray.png' in the '../Output/' directory.
  If the image is successfully saved, the function prints 'Grayscale Succeed!'; otherwise, it prints 'Grayscale Error!'. The memory allocated for the grayData array is deallocated before the function returns.

- **void Filter::auto_color_bal(Image img**)
  Automatically balance the color of the given image and save it to a new file.

@param
---
img: The input image object to be color balanced.

@return

None.

The function takes an Image object and automatically balances the color of the image using the following steps:
First, the function creates a copy of the input image data in a new unsigned char array.
Next, the function calculates the sum of R, G, and B values for all pixels in the image to create a histogram of color values.
Then, the function computes the cumulative distribution function (CDF) of the color values for each channel.
Finally, the function adjusts the R, G, and B values of each pixel in the image by multiplying
the ratio of the CDF of each channel to a corresponding channel value of each pixel. The resulting
adjusted values are stored in the same array as the copy of the input image data.
The function sets the color channels of the input image to the same value and saves the resulting
image to a new file named 'output_bal.png'

- **void Filter::brightness(Image img, double brightness)**
  Adjusts the brightness of an image and saves the result as a new PNG file.

  @param
  ---
  img The input image object.

  brightness The amount of brightness adjustment to apply to the image.

  A positive value increases the brightness, while a negative value decreases it. A value of 0.0 automatically adjusts the brightness based on the average pixel value.
  The function calculates the average pixel value of the input image and applies the brightness adjustment to each pixel. If the brightness parameter is set to 0.0, the function automatically adjusts the brightness based on the average pixel value, such that the resulting image has a similar overall brightness to the original.
  The function saves the result as a new PNG file with the filename "output_bright.png" in the "Output" directory.

- **void Filter::hist_equal(Image img)**

Performs histogram equalization on an image and saves the result as a new PNG file.

@param
---
img The input image object.

The function calculates the histogram of the input image, computes its cumulative distribution function (CDF), and normalizes the intensity values of each pixel using the CDF. The resulting image has a more balanced distribution of pixel intensities, which can improve its visual appearance and make it easier to process with certain computer vision algorithms.
The function saves the result as a new PNG file with the filename "output_hist.png" in the "Output" directory.

- **void Filter::median_blur(Image img, int kernel)**
  Apply a median filter to an input image.
  This function takes an input image and applies a median filter to it, which replaces each pixel value with the median value of its neighboring pixels within a specified kernel size.

  @param
  ---
  img The input image to be filtered

  kernel The size of the kernel used for the median filter (must be an odd integer)

  @return

  None. The filtered image is saved as a PNG file to the "../Output/output_medianblur.png" directory.

- **void Filter::box_blur(Image img, int kernel)**
  Apply a box blur filter to an input image.
  This function takes an input image and applies a box blur filter to it, which replaces each pixel value with the average value of its neighboring pixels within a specified kernel size.

  @param
  ---
  img The input image to be filtered

  kernel The size of the kernel used for the box blur filter (must be an odd integer)

  @return

None. The filtered image is saved as a PNG file to the "../Output/output_boxblur.png" directory.

- **void Filter::gaussian_blur(Image img, int kernel_size, double sigma)**
  Applies a Gaussian blur filter to the input image using a kernel of specified size and sigma value.

  @param
  ----
  img The input image to be filtered.

  kernel_size The size of the Gaussian kernel to be used.

  sigma The standard deviation of the Gaussian distribution.

  @return

  The output image with the applied Gaussian blur filter.

- **void Filter::Sobel(Image img)**
  Applies the Sobel filter to the input image using a 3x3 kernel.
  The filter computes the gradient of the image intensity at each pixel,
  resulting in an image with edges highlighted.

  @param
  ----
  img: Input image to apply the filter to.

  The function saves the filtered image to a new file named "output_sobel.png"
  in the "Output" directory.
  Note: The function deallocates memory for the output array after saving the
  filtered image.

- **void Filter::Prewitt(Image img)**
  Applies the Prewitt filter to the input image using a 3x3 kernel.
  The filter computes the gradient of the image intensity at each pixel,
  resulting in an image with edges highlighted.

  @param
  ----
  img: Input image to apply the filter to.

  The function saves the filtered image to a new file named "output_prewitt.png"
  in the "Output" directory.

Note: The function deallocates memory for the output array after saving the filtered image.

- **void Filter::Scharr(Image img)**
  Applies the Scharr filter to the input image using a 3x3 kernel.
  The filter computes the gradient of the image intensity at each pixel, resulting in an image with edges highlighted.

  @param
  ---
  img: Input image to apply the filter to.

  The function saves the filtered image to a new file named "output_scharr.png" in the "Output" directory.
  Note: The function deallocates memory for the output array after saving the filtered image.

- **void Filter::Roberts(Image img)**
  Applies the Roberts' Cross filter to the input image using a 2x2 kernel.
  The filter computes the gradient of the image intensity at each pixel, resulting in an image with edges highlighted.

  @param
  ---
  img: Input image to apply the filter to.

  The function saves the filtered image to a new file named "output_roberts.png" in the "Output" directory.
  Note: The function deallocates memory for the output array after saving the filtered image.

2.3 *Slice Class*

- **void set_slice_plane(std::string& slice_plane)**
  Set the slice plane to the specified value.
  @param
  ---
  slice_plane the plane along which to slice the volume, either "x-z" or "y-z".
  @throws
  ---
  std::invalid_argument if the slice plane is invalid.

- **void set_slice_pos(int slice_pos, Volume& v)**
  Set the slice position to the specified value.
  @param

---
slice_pos the position of the slice in the slice plane.
v the volume from which to extract the slice.

@throws
---
std::invalid_argument if the slice position is out of range.

- **std::string get_slice_plane() const**
  Get the slice plane.
  @return
  ---
  the plane along which the volume was sliced.

- **int get_slice_pos() const**
  Get the slice position.
  @return

  The position of the slice in the slice plane.

- **void get_slice_pic(Volume& v)**
  Get the slice picture data.
  @param
  ---
  v the volume from which to extract the slice.

2.4 *Projection Class*

- **void find_z_range(const int& zmin, const int& zmax)**
  Compute correct z range of the user specified thin slab
  @param
  ---
  zmin, zmax: minimum and maximum z specified by user

- **unsigned char* max_ip(int zmin = NULL, int zmax = NULL)**
  Maximum intensity projection
  @param
  ---
  zmin, zmax: minimum and maximum z coordinates of the user specified thin slab
  @return
  ---
  data: output projection data

- **unsigned char* min_ip(int zmin = NULL, int zmax = NULL)**
  Minimum intensity projection
  @param
  ---
  zmin, zmax: minimum and maximum z coordinates of the user specified thin slab
  @return
  ---
  data: output projection data

- **unsigned char* avg_ip(int zmin = NULL, int zmax = NULL)**
  Average intensity projection
  @param
  ---
  zmin, zmax: minimum and maximum z coordinates of the user specified thin slab
  @return
  ---
  data: output projection data

2.5 *Volume Class*

- **void get_slice_pic(Volume& v)**
  Get the slice picture data.
  @param
  ---
  v the volume from which to extract the slice.
- **int get_width()**
  Returns the width of the volume.
  @return
  ---
  the width of the volume

- **int get_height()**
  Returns the height of the volume.
  @return
  ---
  the height of the volume

- **int get_channel()**
  Returns the number of channels in the volume.
  @return
  ---
  the number of channels in the volume

- **void set_height(int h)**
  Sets the height of the volume.

@param
----
h the new height value

3. USER INTERFACE

To execute the program, the user needs to run the “main_2D.cpp” or “main_3D.cpp” file based on the users’ requirements.
Once the user runs the main_2D program, they will be asked to provide a path for the image they want to apply filters on. The images will get stored in the Output folder.
The following list gets displayed to the user:
“””
1. Automatic Color Balance
2. Histogram Equalisation
3. Grayscale
4. Brightness
5. Median Blur
6. Box Blur
7. Gaussian Blur
8. Sobel
9. Prewitt
10.Scharr
11. Roberts

“””
The user will be prompted to select an option to apply a filter on the image the user has provided. In case the user does not provide an image, an image will be set by default and the chosen filter will be applied on the image and the output image will be stored in the Output folder.

The user will be allowed to select filters repeatedly until they type ‘n’ to terminate the program.

Similarly, if the user runs the main_3D program, will execute the 3D filters and the Slicing class.

4. OUTPUT

All output images are saved in the Output folder.

5. TESTING

To perform testing on the classes and their functions, a class Tests is defined. It is in the Tests/ directory. To run unit tests on the project, tests.cpp file should be executed. The following functions are defined in the Tests class.

- **void Tests::test_automatic_color_balance()**

Test the automatic color balance function by comparing the output image with the expected image.
@param

None

@return

None

- **void Tests::test_grayscale()**
  Test the grayscale function by comparing the output image's expected channel, height, width and pixel value ranges
  @param

  None

  @return

  None

- **void Tests::test_brightness()**
  Test the brightness function by comparing the output image with the expected image
  @param

  None

  @return

  None

- **void Tests::test_median_blur()**
  Test the median blur function by comparing the output image with the expected image after applying different kernel sizes.

  @param

  None

  @return

  None

- **void Tests::test_hist_equal()**
  Test the histogram equalization function by comparing the output image with the expected image.

  @param

  None

  @return

  None

- **void Tests::test_box_blur()**
  Test the box blur function by comparing the output image with the expected image.

  @param

  None

  @return

  None

- **void Tests::test_gaussian_blur()**
  Test the gaussian blur function by comparing the output image with the expected image.

  @param

  None

  @return

  None

- **void Tests::test_sobel()**
  Test the Sobel edge detection function by comparing the output image with the expected image.

  @param

None

@return

None

- **void Tests::test_prewitt()**
  Test the Prewitt edge detection function by comparing the output image with the expected image.

  @param

  None

  @return

  None
- **void Tests::test_fracture()**
  @param

  None

  @return

  None

- **void Tests::test_scahrr()**
  Test the Scharr edge detection function by comparing the output image with the expected image.

  @param

  None

  @return

  None

- **void Tests::test_roberts()**
  Test the Roberts edge detection function by comparing the output image with the expected image.

  @param

None

@return

None

- **void Tests::test_confuciusornis()**
  @param

  None

  @return

  None

6. REFERENCES